NCPU：迈向受细胞启发的空间计算

Iliya Zhechev 索非亚大学，
保加利亚
iliya.zhechev@gmail.com

Piotr Walas 华沙理工大学，波兰
Piotr.Walas@pw.edu.pl

arXiv:2611.32750v1 [cs.NE] 24 Nov 2026

摘要—**简单的元胞自动机（CA）可以实现计算通用性，但其规则是手工设计的。我们通过梯度下降训练神经元细胞自动机（NCA），使其学习局部更新规则，从而计算精确的数字逻辑。位被编码为二维网格上的单元簇，结果只有通过中间单元传播后才能到达输出端。每项任务分别训练一个模型：逻辑门、4位和8位脉动进位加法器，以及初步的8位算术逻辑单元（ALU）。所有逻辑门和加法器模型在10个随机种子中的10个都达到**100% **的位正确率；对所有输入对进行穷举测试时，4位情况下最多出现256个输入对中的1个错误，8位情况下最多出现** 65536 **个输入对中的7个错误。在最多3位操作数上训练的加法器可以泛化到8位。加法器以波的形式传播进位，进位链中每增加1位，延迟约为3.5步。只有在训练滚动长度内，输出才是正确的。**

索引词—**神经元细胞自动机、演化硬件、空间计算、数字逻辑**

I. 引言

通过梯度下降学习得到的一条规则，在网格的每个单元上以完全相同的方式应用，并且只能观察其邻居，它能计算精确的数字逻辑吗？元胞自动机（CA）是一种离散动力系统，其中一条局部规则更新网格中的每个单元。简单的元胞自动机（CA），例如康威生命游戏 [1]，具有计算通用性，但其规则以及使其能够进行计算的初始状态，都是手工设计的。神经元细胞自动机（NCA） [2] 用一个根据初始网格状态和目标网格状态训练的神经网络取代固定规则；训练得到的模型像元胞自动机一样更新网格。

大多数神经元细胞自动机工作都针对模式生长和分类，在这些任务中，输出的每个区域主要取决于附近的单元。在数字逻辑中，一个输出位可能取决于网格上相距很远的输入位。加法就是一个简单例子：进位可以经过操作数的每一位传播。我们训练的神经元细胞自动机以一种类似行波进位加法器的方式解决这一问题：进位像波一样从一位移动到另一位。波的每一步都是相同的局部运算，因此这些模型也能对宽度超过训练样本的操作数进行加法。

我们将神经元细胞自动机（NCA）作为神经元胞处理单元（NCPU）进行研究。这是一种经过训练、能够以空间方式执行数字逻辑的基底。我们的目标并非将所有计算都转移到 NCPU 上，而是要探究 NCA 空间计算是否可行，以及实现它需要什么条件。我们针对一系列复杂度逐步增加的任务分别训练 NCA 模型，从逻辑门到行波进位加法器，再到初步的算术逻辑单元（ALU）；所有输入和输出都编码为二维网格上的空间比特模式，并通过消融研究考察模型超参数对动态过程的影响。

这个问题有两方面的重要意义。对于人工生命而言，精确逻辑是检验自组织局部规则能力的一项严格测试：一个错误就会翻转一位，因此成功与失败泾渭分明；而规则所发现的机制，例如第VI-B节中的进位波，则展示了全局函数如何由局部交互产生。对于非常规计算而言，能够实现算术原语的 NCA，是迈向由相同局部单元构成的可训练硬件的一步。

我们的贡献如下：（1）基于神经元细胞自动机的数字计算，涵盖从单个逻辑门到8位加法器的一系列任务，每项任务使用一个模型，并实现了一个初步的算术逻辑单元；（2）从较小加法器到较大加法器的长度泛化；（3）一种由空间传播约束产生的进位波机制，并通过其传播延迟和通道消融进行测量。本研究的代码和数据可在 https://github.com/ichko/ncpu 获取，交互式演示可在 https://ncpu.pages.dev 查看。

II. 相关工作

学习到的更新规则从多个方向扩展了经典元胞自动机。神经元细胞自动机由谷歌的一个研究团队提出，他们展示了学习到的规则能够维持并再生受损模式 [2]；后续研究表明，神经元细胞自动机可以泛化到矩阵计算，提示其可成为通用计算的构建模块 [3]。其他研究通过质量守恒 [4]，、异步性 [5]和各向同性 [6]，来控制神经元细胞自动机的动态，而可微分逻辑元胞自动机则连接了基于规则的自动机与学习式模式生成 [7]。神经元细胞自动机已应用于MNIST分类 [8]，、信号响应 [9],和分布式感知 [10]与物理二维形状分类 [11]。

神经元细胞自动机已在嵌入式芯片 [12],上实现，同时还探索了忆阻器 [13]和光子 [14] 基底，这表明类似 NCPU 的单元原则上可以在低功耗硬件中实现。

相较于 Béna等人 [3]，研究神经元细胞自动机能否近似通用计算，我们的贡献具有互补性：我们并不追求通用性，而是探讨神经元细胞自动机的动态是否能够实现精确的行波进位算术，以及这会诱导出哪些涌现机制。

与我们工作最接近的是，Wei [15] 从局部性、对称性和稳定性约束出发，推导出一种神经元细胞自动机架构（SEAD），不断迭代局部卷积规则，直至其收敛到一个吸引子，并报告了在奇偶校验、多位加法以及规则110学习上的长度泛化。该工作集中于单一运算的长度泛化。而我们则采用标准神经元胞自动机公式 [2]，并探究它能够实现哪些

发表于2027年IEEE计算智能研讨会系列（SSCI 2027）。© 2027 IEEE。允许个人使用本材料。对于所有其他用途，包括在任何当前或未来的媒体中重印/再版本材料、将本材料用于广告或宣传目的、创作新的合集作品、转售或再分发至服务器或列表，或在其他作品中重复使用本作品的任何受版权保护的组成部分，必须获得IEEE的许可。

数字构建模块：逻辑门、带显式进位输出的脉动进位加法器，以及操作码控制的算术逻辑单元；其中，位编码为空间簇，输入和输出位于二维网格的不同区域。我们还分析了训练后的加法器所采用的机制（进位波），以及该方法失效的地方（算术逻辑单元的进位输出）。

III. 方法

A. 设置

NCPU 模型流程始于感知阶段，在该阶段中对当前网格状态应用 Sobel 滤波。随后，将所得的感知状态传递至 *NCA* 更新阶段，在此应用学习到的 NCA 转移规则。接着进行存活掩码 步骤，将状态值低于预定义阈值的单元停用。最后，为促进泛化，在随机更新阶段中对状态更新进行随机掩码处理，即每个单元的更新以固定概率应用，该概率称为激活概率。

全部四个阶段均遵循标准神经元胞自动机公式 [2],，但有一处例外：我们禁用了存活掩码。第 IV 节表明，掩码会阻碍这些任务的收敛，因此下文报告的每项实验均在关闭该功能的情况下运行（存活掩码设为 0.0）。

此外，所有实验均使用 16 个通道和标准的学习到的转移网络 [2], ，即一个包含 128 个单元的单隐藏层，该网络应用于三个基于 Sobel 的感知通道；加法器和算术逻辑单元实验使用激活概率 0.5 以及 7×7 卷积核，门实验则使用 5×5 （第 IV 节报告了促使我们采用更大卷积核的卷积核扫描结果）。

优化使用Adam，学习率为 10^{-4}，批大小为 8（逻辑门实验中为 16）；每个优化步均匀采样展开长度：逻辑门为 32–64 步，加法器和长度泛化为 64–128 步，ALU 为 48–96 步。

每项实验中的网格均保持固定：输入和输出位簇（第 III-B 节）被放置在固定的空间布局中，随着位宽增大，该布局会在网格中容纳更多位簇，而不是调整网格大小；同时，布局采用底部对齐，使最低有效位（LSB）在每个位宽下都占据相同的绝对位置。网格大小根据任务分别选定，以便在实际内存约束下容纳其必须支持的最大编码：28×48 用于单个逻辑门，80×112 用于 4 位加法器（两种布局）、双列 8位加法器以及所有长度泛化运行，80×192 用于单列 8位加法器，96×112 用于 ALU。

B. 实验设置

NCPU 在一个二维网格上运行，网格单元值处于双极范围 $[-1, 1]$ 内，因此逻辑低电平和逻辑高电平分别位于相反的极值（分别为 -1 和 $+1$ ），而死亡细胞保持为 0，从而避免单极 $[0, 1]$ 范围所引入的偏置。

一个比特并非由单个单元承载，而是由一个位簇承载：一个半径为 4 个单元的圆盘，其中每个单元都保存相同的值，$+1$ 或 -1。比特的读取方式是取圆盘内平均激活值的符号因此，单个单元中的噪声不会改变该值，而位簇则为局部规则提供了一片可供信号路由的区域。

为确保计算传播，网格被划分为输入区域（左侧）和输出区域（右侧），因此模型必须将信息从左向右传播并进行转换。初始状态仅包含输入比特，而目标状态仅定义输出位。比特的数量和排列方式因任务而异，范围从逻辑门到算术逻辑单元。

C. 损失函数

对于 NCPU，我们使用**掩码均方误差（MSE）损失**，在选定的网格单元和展开帧上进行计算，从而能够在训练期间对输出区域实施有针对性的监督。

$$\mathcal{L}_{\text{masked}} = \frac{1}{B\,|\mathcal{T}|\,\sum_i m_i} \sum_{b=1}^{B} \sum_{t \in \mathcal{T}} \sum_i m_i \left(\hat{y}_{b,t,i} - y_{b,i}\right)^2 \quad (1)$$

目标 y 是一个与状态大小相同的网格。在每个输出位的圆盘内，如果该位为 1，则每个单元都设为 $+1$ （实现中为 0.99，取自 8 位像素值）；如果该位为 0，则设为 -1 。输出圆盘之外的所有单元均设为 0，并具有 $m_i = 0$，因此不会受到监督。输入比特以相同方式写入初始状态中各自的输入圆盘内，写入通道0和通道1；其他所有单元和通道均从 0 开始。通道1为只读，因此输入在整个展开过程中始终可见；而同样承载输出的通道0则可以自由变化。

在损失函数中，b 索引批次中的 B 个样本，t 索引受监督的展开帧 $\mathcal{T}$ （对于门和加法器实验，包含展开过程的每一帧；对于 ALU，包含最后八帧），i索引 N 个网格单元，$\hat{y}_{b,\,t,\,i}$ 表示单元 i 在帧 t的通道 0 激活值，$y_{b,\,i}$ 表示单元 i的目标激活值，而 $m_i \in \{0, 1\}$ 当且仅当单元 i 位于输出区域（即某个位簇内）时取值为 1。由于 $|\mathcal{T}|$ 因任务类别而异，损失值和收敛步数在同一类别内具有可比性，但不同类别之间不能直接比较。

所有内部动态，包括第 VI-B 节中的进位波，均不接受直接监督。

D. 指标

NCPU 模型根据以下指标进行评估。**位正确率** 是与目标相符的输出位所占比例，并在批次上取均值。如果最终展开帧 T上某个输出簇的通道 0 均值激活值的符号与目标位一致（逻辑 1 为正，逻辑 0 为负），则该位计为正确。测试输入从各任务的有效输入中均匀随机抽取。对于 ALU，每种运算有 64 个样本，因此单比特准确率的 95% 置信区间约为 ± 12.5 个百分点。每次评估都会从测试输入开始，运行一次包含 T 步的全新展开；每个测试输入仅使用一次。

收敛速度是达到给定比特正确率阈值（除非另有说明，阈值为 100%）所需的训练步数；每项任务使用 10 个随机种子进行测量，并报告均值 $\pm$标准差。

IV. 消融

A. 收敛空间扫描

为选择超参数，我们针对单门任务，对卷积核大小、存活掩码阈值和权重初始化进行了网格搜索，并测量归一化收敛速度。在消融实验中，我们将模型达到 100% 位正确率视为收敛。收敛速度采用从 1.0 到 0.0 的反向刻度表示，其中 1.0 表示在前 2000 步内收敛，0.0 表示未能收敛。

该模型对存活掩码尤为敏感：禁用存活掩码时可获得最佳收敛效果，而启用存活掩码时，成功运行的次数几乎为零。这是一个负面结果：它与最初的 Growing NCA 表述相矛盾 [2],，后者认为掩码能够稳定模式生长；而在这里，掩码会停用从左向右传播必须经过的中间单元。因此，我们在后续所有实验中都将其禁用。该扫描使用单门任务；我们预计，随着电路深度增加，这种影响会更加明显，因为掩码会移除较长路径（例如 8 位进位链）必须跨越的近零中间单元。

核大小的影响力排名第二：更大的核始终会缩短收敛时间，这表明更宽的感受野能够为每个单元提供更多上下文。核大小为 7 是收敛速度与计算成本之间的实用折中方案，我们在加法器和算术逻辑单元实验中采用该设置；第五节的门运行使用 5×5 训练器默认设置。

该模型也更倾向于采用随机权重初始化，而不是零权重初始化；较高的激活概率往往也有所帮助，这对于更复杂的电路可能很重要。

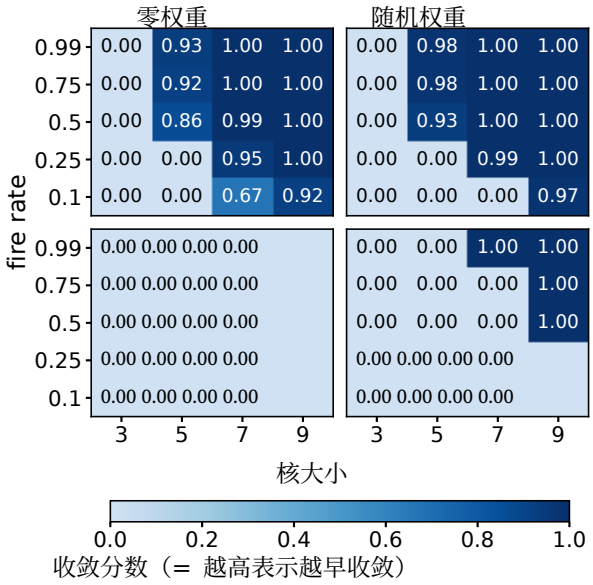


图1：当存活掩码阈值为 0.0 和 0.1 时（按行；阈值为 0.3 时没有运行收敛），不同权重初始化（按列）、激活概率和核大小下单门任务的收敛情况。蓝色越深表示收敛越快：1.0 表示在前 2 000 步内收敛，0.0 表示未收敛。

V. 结果

我们在一系列复杂度递增的任务上评估 NCPU，每个任务分别训练一个模型：单个逻辑门、脉动进位加法器、长度泛化，以及初步的 8位算术逻辑单元。所有实验均使用掩码均方误差损失（第 III-C 节）。

A. 逻辑门

我们在六种标准双输入门（与、或、或非、与非、异或、同或门）以及两种稍复杂的电路（半加器和多数门）上训练了 NCPU。每项实验使用 10 个随机种子进行训练；收敛速度和比特准确率报告为跨随机种子的均值 ±标准差。

表I：逻辑门、加法器和算术逻辑单元实验的多随机种子收敛结果。收敛步数是首次达到 100% 位正确率所需的优化步数。每个门和加法器的数值均为每项任务 10 个随机种子的均值，“±”表示这些随机种子之间的总体标准差（ddof= 0）；算术逻辑单元是一次单独的未收敛运行，未按随机种子取平均。最终比特是最后一个展开帧中的正确输出位；根据任务不同，最大值分别为 1（逻辑门、三输入多数门）、2（半加器）、5（4 位）或 9（8 位）。*cols1* 和*cols2* 表示第五节 B 中的单列和双列操作数布局；4 位 cols2 和 8 位 cols1 行是布局对照，训练了 20000 步。

Task	随机种子	收敛步数	最终比特
AND	10	49 ± 35	1.00 ± 0.00
NAND	10	45 ± 21	1.00 ± 0.00
NOR	10	35 ± 12	1.00 ± 0.00
OR	10	54 ± 36	1.00 ± 0.00
XNOR	10	67 ± 23	1.00 ± 0.00
XOR	10	85 ± 42	1.00 ± 0.00
半加器	10	366 ± 138	2.00 ± 0.10
三输入多数门	10	116 ± 53	1.00 ± 0.00
4位加法器（cols1）	10	9 976 ± 1 342	5.00 ± 0.00
4位加法器 (cols2)	10	3 770 ± 824	5.00 ± 0.00
8位加法器 (cols2)	10	5 142 ± 1 035	9.00 ± 0.00
8位加法器 (cols1)	10	未收敛（20 000 步数）	
算术逻辑单元 (8位)	1	未收敛（600 000 步数）	

线性可分门（与、与非、或非、或）在不到55步内收敛，非线性可分门（异或、同或门）在67–85步内收敛，而三输入多数门约需116步；半加器必须同时输出和与进位，约需366步。不同随机种子下也呈现出相同的排序（图2）。加法器也都完全收敛（5/5位和9/9位，表I），但需要多得多的优化步数；算术逻辑单元在600000步后仍未收敛。

与门和半加器的示例演化过程（图3）显示出清晰的传播路径。它们还显示，每个网格左侧都有一片高度混沌的区域。我们将此视为基于波的计算的迹象，但尚未验证这一解释，因此仅将其作为观察结果报告。

B. 脉动进位加法器

我们在两个规模（4位和8位，每种10个随机种子）上训练了脉动进位加法，生成和与进位输出。

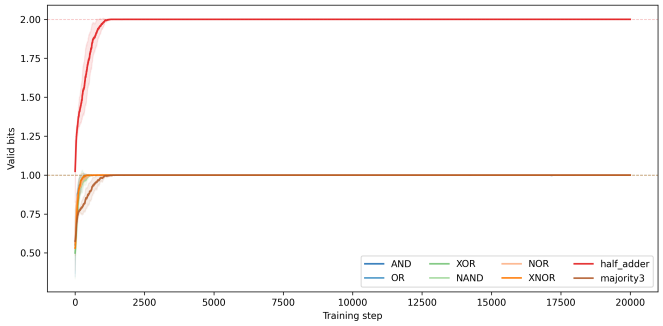


图2：所有逻辑门、半加器和三输入多数门在不同随机种子下的训练准确率（正确输出位）。线性可分门比非线性门收敛更快。

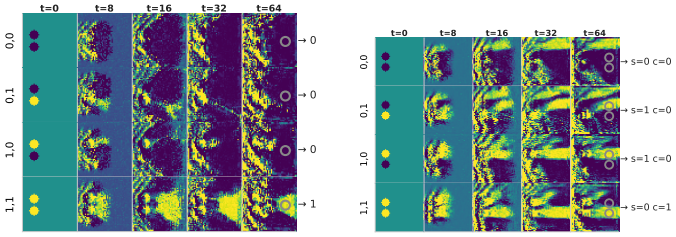
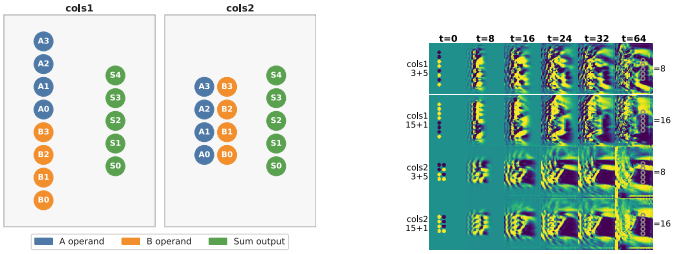


图3：与门（左）和半加器（右）的通道0展开。圆圈表示输出位位置。左侧的混沌区域可能表明存在基于波的计算（未经测试）。

4位情形比较了单列布局（*cols1*）和双列布局（*cols2*），其中操作数 A 和 B 位于相邻列中（图4a）。cols2 在 $3\,770 \pm 824$ 步内收敛，比 cols1（$9\,976 \pm 1\,342$）快2.6倍；通道0展开轨迹（图4b）显示，cols2 更早稳定其输出列。双列8位加法器在 $5\,142 \pm 1\,035$ 步内收敛。为区分布局与操作数位宽的影响，我们还在单列布局中训练了8位加法器：10 个随机种子均未能在20 000步内收敛（最佳结果为9位中的6.4至7.0位；其中一个随机种子运行至50 000步后达到6.75位）。因此，在位宽固定时，布局决定收敛情况。



（a）输入布局　　（b）通道0展开轨迹

图4：4位加法器：单列（*cols1*）与双列（*cols2*）输入布局（a），以及两种布局在相同两个输入组合上的通道0展开（b）。前两行显示 cols1；后两行显示 cols2。

C. 精确性

训练期间的位正确率通过随机输入进行测量。为测试精确性，我们在第 128 步对每个输入对评估了训练后的加法器，使用的是采用相同代码重新训练的模型：十个 4 位 cols2 模型、三个 4 位 cols1 模型和两个 8 位 cols2 模型。13 个 4 位模型中有 11 个是正确的在全部 256 个输入对上均正确，另外两个模型各有一个输入对出错；两个 8 位模型分别在 65 536 个输入对中的 7 个和 6 个上出错（正确率为 99.99%）。当训练中使用的展开范围（从第 64 步到第 128 步）内的每一步都必须正确时，结果也相同，但另有 3 个 8 位输入对除外。超出该范围后，答案并不稳定：运行至 1 024 步时，测试的 4 个模型中有 3 个在第 256 步前丢失了大部分答案（仍有 2–69% 正确），第四个模型则一直到第 1 024 步仍保留一半答案。这些模型能在训练时间范围内计算出和，但无法将其保持为不动点。

D. 长度泛化

为了测试模型能否在较小的加法器上训练，并在训练期间未接触较大加法器的情况下泛化到更大的加法器，我们在固定的 80×112 网格上使用 K 位加法器（$K \in \{2, 3, 4, 5, 6\}$）训练模型，随后在 $N \geq 1$ 条件下评估其在测试位宽 $K + N$ 上的表现。位簇从网格底部向上放置，每种位宽下的位置都相同。在 K 位上训练的模型只使用网格的下半部分。测试时，我们在已训练的位上方添加位，并检查学到的规则能否扩展到这些位。在 $K \geq 3$ 位上训练的模型能够近乎完美地外推至所有不超过 8 位的宽度，无论差距如何，位正确率均至少达到 98%。仅用 $K = 2$ 位训练仍然有效，但性能会随差距增大而按比例下降。在 $K = 3$ 处可以看到明显的转折：低于该值时，泛化能力随差距增大而下降；高于该值时，准确率不受差距影响（图 5）。

我们推测，尽管尚未进行直接测试，$K = 3$ 之所以重要，是因为它是 LSB 与最高有效位（MSB）之间包含一个中间位的最小位宽。在$K = 2$ 的情况下，模型从未将进位传播经过中间位置，因此测试时新增的位置对它而言是全新的；而在 $K \geq 3$ 的情况下，它已经学会了这一步，将更多位置打包到网格中会再次应用相同的局部规则。

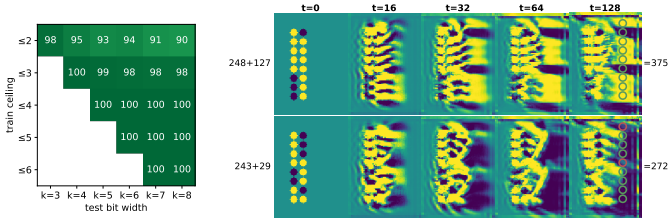


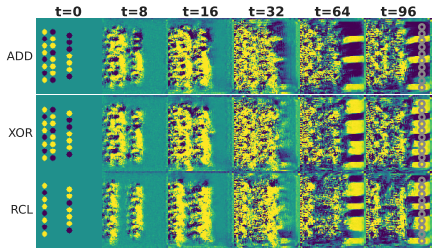
图 5：长度泛化结果。左：每个训练上限 K 和测试位宽对应的准确率（%）。右：8 位输入下，K=3 模型的成功（上）与失败（下）。

E. 算术与逻辑单元

算术逻辑单元实验尚处于初步阶段：这是一次包含 600000 步优化步数的单次运行；在我们的计算预算内，无法使用其他随机种子重复该实验。模型未达到 100% 的位正确率。我们报告此次运行，是因为它展示了该方法在哪些方面有效、又在哪些方面失效；但下文关于算术逻辑单元的所有发现仅适用于这一次运行，并非收敛结果，也不是对多个随机种子取平均后的结果。该布局使用两个输入列分别承载操作数 A和 B，一个控制列承载 3 位操作码、输入进位和 3位条件码，右侧则设有一个



（a）8位算术逻辑单元的空间布局网格。



（b）加法（进位密集）、异或（按位运算）和带进位循环左移（续）的通道0展开。灰色圆圈标记输出位位置。

图6：8位算术逻辑单元实验：空间布局（a）以及三种代表性运算的通道0展开（b）。

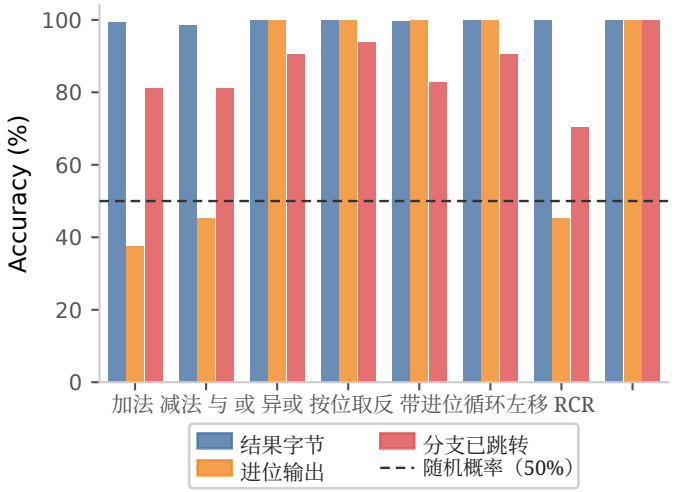


图7：按输出组划分的8位算术逻辑单元各操作准确率：结果位、进位输出、分支已跳转。单次未收敛运行；随机概率基准的注意事项见第V-E节。虚线表示单个位的50%随机概率基准。准确率缺口集中在加法、减法和带进位循环左移的进位输出上。

包含十个位：8位结果、进位输出和分支已跳转（图 6a 和图 6b）。

算术逻辑单元支持八种运算：加法（$A + B + c_{\text{in}}$）、减法（$A - B - c_{\text{in}}$；进位输出为 $=$ 按位取反借位）、与、或、异或、按位取反（对 B）、带进位循环左移（带进位循环左移）以及 RCR（带进位循环右移）。对于加法和减法，进位输出是算术进位；对于带进位循环左移和RCR，则是从操作数中移出的位；对于按位运算，其定义为零。计算结果以及内部表示的零/负/进位标志满足3位条件码（EQ、NE、CS、CC、MI、PL、AL、NV）时，分支已跳转标志被置位。因此，该规则需要对自身表示的标志执行谓词求值，而不是输出一个存储位。

尽管所有运算的结果位准确率都近乎完美（每种运算至少为 98.6 %，均值为 99.7 %），但辅助输出揭示了具有结构性的失效模式（图 7）。按输出分项分析可以定位准确率差距。进位输出只有一位，因此随机概率为 50%；对于加法、减法和带进位循环左移，进位准确率为 37–45 %；每种运算采样 64 次时，95 % 置信区间约为 ± 12.5 个百分点，因此模型并未学会这些运算中的进位，我们也不声称其表现低于随机概率。对于与、或、异或和按位取反，进位输出定义为零，因此它们达到 100 %，也可以由永不输出进位的规则实现；只有 RCR 的进位输出是真正的 50/50 位（即 B 的最低有效位），因此具有信息量，而该运算的准确率达到 100 %。因此，在这次运行中，模型只学会了 RCR 的进位输出，未学会三种运算（加法、减法和带进位循环左移）的进位输出；其余四种运算则按定义不具信息量。分支执行准确率平均为 86.3%，高于 50 % 的随机基线，但远低于结果位准确率的上限。带进位循环左移和 RCR 的区别仅在于复制到进位输出的操作数位不同（最高有效位与最低有效位），但二者在这一表现划分中恰好分属两侧；由于只有一次未收敛运行，我们无法判断这种不对称是结构性的，还是运行中的偶然现象，因此将其列为待解问题。

VI. 讨论

A. 基线与范围

我们将这项工作定位为一项存在性证明。我们要探究的是：在传统网络不面临的约束下，是否能够训练一个纯局部、统一的神经元细胞自动机更新规则来执行结构化数字逻辑：具有 7×7 邻域、没有全局寻址，并且执行原位计算，在此过程中，信息只能通过中间单元的传播到达输出。现有已发表的工作中没有可直接比较的基线：大多数先前的神经元细胞自动机工作以模式生成为目标，而最接近的工作 [15] 研究的是采用不同神经元细胞自动机架构的加法长度泛化。具有相同 7×7 感受野的前馈卷积神经网络（CNN）需要为每个传播步设置一层，并使用多得多的参数，才能到达输出；我们关注的问题不是某个网络能否执行加法，而是所有单元共享的一条局部规则能否做到这一点，以及这种规则会诱导出怎样的动态。相关的控制因素是内部因素：上文的核大小和布局消融实验，以及机会水平分析。

B. 进位波

尽管 NCA 的更新规则是局部的，且以统一方式应用，但这个 8 位加法器学会了一种顺序式 策略，让人想起经典的行波进位加法器。在 $127 + 1 = 128$ 上运行训练好的模型——其进位链有 7 位——并检查隐藏通道，可以发现一个波前：它从最低有效位输出附近开始，在数十个时间步内向上扫过，直至进位输出的最高有效位（图 8）。在下方的进位信号到达之前，高位实际上会保持中性；这与行波进位电路在硅中施加的约束相同，然而训练损失中并不存在时间结构。这种模式与输出簇之间的空间距离会带来信息传播所需的最少更新次数这一点相符。

我们测量了两个 8 位模型和一个 4 位模型中的波。对于每个输出位，我们记录它从哪一步开始始终保持正确，并根据以该位为终点的进位链长度对各位进行分组（128 个随机输入和最坏情况进位链）。稳定所需的步数随进位链长度线性增长：没有进位时为 6–9 步，8 位进位链则为 33–35 步。进位链上的每一位约增加 3.5 步（两个 8 位模型分别为 2.6 步和 3.5 步，4 位模型为 4.4 步），这与进位从一个位簇传递到下一个位簇时预期的延迟相符。为了测试

波是否具有因果性，我们在每一步都将一个隐藏通道设为零。大多数通道对每次加法都必不可少，但每个模型中都有一个通道专门负责进位。对于第一个 8 位模型，移除通道 5 后，不含进位的加法中有 90 % 仍然正确，而进位链长度为三位或以上的加法中只有 18 % 正确；对于第二个 8 位模型的通道 11，这两个比率分别为 90 % 和 0 %；对于 4 位模型的通道 5，则分别为 88 % 和 0 %。移除该通道会恰好破坏那些需要进位的加法。8 位模型的测试输入中只有 10 个不含进位，因此这些比率的精度较低。

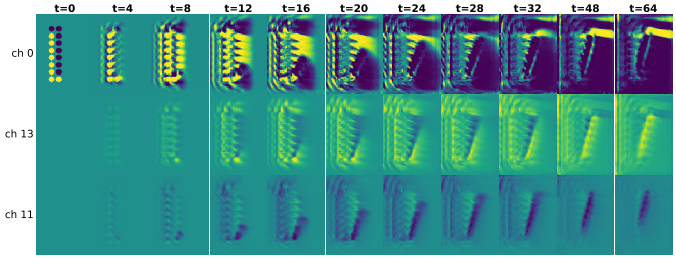


图 8：在加法 $127 + 1 = 128$（7 位进位链）期间最强烈表达进位波的隐藏通道。每一行表示经过训练的 8 位加法器中的一个隐藏通道；各列表示 NCA 时间步。波前从最低有效位（底部）向进位输出最高有效位（顶部）传播，历时约 30 步。

C. 空间布局效应

在所有多位实验中，在空间布局中将相互作用的位放置得更近，都会持续缩短收敛时间。在加法器实验中，双列布局优于单列布局，即使在宽度固定时也是如此（第五节 B）。我们的解释（这是一种假设）是，在双列布局中，相应的操作数位在空间上彼此相邻。这可能会体现出增大感知核大小的效果：两者都会增加单个时间步内每个单元可获得的相关上下文信息量，从而减少在网格中传播信息所需的步数。

D. 算术逻辑单元限制与进位位失效（单 *ru n)*

在我们的单次运行中，ALU 在八种运算中的三种上无法正确产生进位输出。以下解释仅是一种假设。ADD 和 SUB 需要持久的进位链；我们的假设是，在操作码条件控制下，纯加法器模型（图 8）的进位波前更难以维持，因为波前必须在各个位位置之间保持激活，同时网格的其余部分还要为选定的操作码路由操作数数据。根据我们的定义，RCL 不需要进位链：其进位输出只是从操作数复制而来的单个位，因此这一解释无法涵盖 RCL；对于 RCL 失败而其镜像操作 RCR 成功，我们也没有解释。要解决进位输出问题，需要更长的训练预算、保持进位的通道分配或更丰富的感知核，同时还需要更大的评估样本；我们将其留待未来工作。

VII. 结论

我们已经证明，在纯粹的局部统一更新规则下，神经元细胞自动机能够经过训练，执行结构化的数字计算，涵盖从单个逻辑门到带有初步算术逻辑单元的 8 位脉动进位加法器这一系列任务。

收敛结果（除非另有说明，否则使用 10 个随机种子）：(1) 每个门和加法器模型均达到 100% 的位正确率，穷举测试在 65 536 个输入对中最多发现 7 个错误；(2) 在位宽固定时，操作数布局决定收敛；(3) 使用 $K \geq 3$ 位进行训练的加法器可以外推到 8 位；(4) 进位波在进位链的每一位上都有可测量的延迟，并且依赖于特定的隐藏通道。

局限性与假设：答案在训练滚动长度范围内是正确的，但超出该范围后会变差。$K = 3$在长度泛化中的作用以及操作数邻接性的影响都只是一些假设。ALU 仅进行了一次运行，尚未收敛，而且只学会了 RCR 的进位输出。

接下来的两个步骤是：对 ALU 进行多随机种子评估，以及训练模型，使其输出在超出训练时域后仍保持稳定。

致谢

本文在 AI 工具的协助下完成：使用了 Anthropic 的 Claude（多个模型版本） [16] 以及 OpenCode 编码代理 [17], 进行语言润色、格式整理和代码辅助。作者审阅了所有由 AI 辅助生成的文本和代码；实验结果、分析与结论均为作者原创。本研究的代码和数据可在 https://github.com/ichko/ncpu 获取，交互式演示可访问 https://ncpu.pages.dev。

参考文献

[1] M. Gardner, 《数学游戏》，《科学美国人》，第 223 卷，第 4 期，第 120–123 页，1970 年。 [在线]。网址：http://www.jstor.org/ stable/24927642[2] A. Mordvintsev、E. Randazzo、E. Niklasson 和 M. Levin, 《生长型神经元细胞自动机》，*Distill*，第 5 卷，第 2 期，2020 年。 [在线]。网址：https://distill.pub/2020/growing-ca [3] G. Béna、M. Faldor、D. F. M. Goodman 和 A. Cully, 《通向通用神经元细胞自动机之路》，2025 年。 [在线]。网址：https://arxiv.org/abs/ 2505 13058 。V. Papadopoulos 和 E. Guichard, 《Mace：元胞自动机的通用质量守恒动力学》，2025 年。 [在线]。网址：https: //arxiv.org/abs/ 2507 12306 。E. Niklasson、A. Mordvintsev 和 E. Randazzo, 《神经元细胞自动机中的异步性》，载于*ALIFE 2021*：*2021* 年人工生命会议，2021 年 7 月，第 116 页。 [在线]。网址：https: // doi.org/ 10 1162 _ _00461 。/isal a A. Mordvintsev、E. Randazzo 和 C. Fouts, 《生长型各向同性神经元细胞自动机》，2022 年。 [在线]。网址：https://arxiv.org/abs/2205. 01681[7]P. Miotti、E. Niklasson、E. Randazzo 和 A. Mordvintsev, 《可微逻辑细胞自动机：从生命游戏到模式生成》，2025 [ ] 2506 04912 。在线。网址：https://arxiv.org/abs/。E. Randazzo、A. Mordvintsev、E. Niklasson、M. Levin 和 S. Greydanus, 《自分类 MNIST 数字》，*Distill*，第 5, 8, 卷，第 期。在线。2020 [ ] 2020 网址：https://distill.pub/ /selforg/mnist [9] J. Stovold, 《神经元细胞自动机能够响应信号》，载于*2023* 年人工生命会议。MIT 出版社，2023 年。 [在线]。网址：https://doi.org/ 10 1162 _ _00567 。/isal a B. Dacre、N. Bessone、M. L. Preti、D. Cafiso、R. Moreno、A. Faíña 和 L. Beccai, 《使用软式电感传感器阵列实现分布式操纵器系统去中心化感知的神经元细胞自动机》，载于*2025* 年第 *11* 届国际控制、自动化与机器人会议（*ICCAR*）。IEEE，2025 年 4 月，第 314–321 页。[11] K. Walker、R. B. Palm、R. M. Garcia、A. Faina、K. Stoy 和 S. Risi, 《用于二维形状分类的物理神经元细胞自动机》，2022 年。[在线]。网址：https://arxiv.org/abs/2203.07548

[12] D. Woiwode、J. Marten 和 B. Rosenhahn，《用于（神经）元胞自动机的旋转不变嵌入式平台》，2025 年。 [在线]。网址：https://arxiv.org/abs/2510 07440 。H. Wang、J. Wang、S. Yan、R. Pan、M. Sun、Q. Yu、T. Chen、L. Chen 和 Y. Liu，《通过状态化三忆阻器逻辑运算实现初等元胞自动机》，《科学报告》，第 14, 1, 2677, 卷，第 期，2024 [ ] 10 1038 41598 024 53125 第 页。在线。网址：https://doi.org/ 。/s - - -w [14] G. H. Y. Li、C. R. Leefmans、J. Williams 和 A. Marandi，《用于复杂现象模拟的光子初等元胞自动机》，《光：科学与应用》，第 12 卷，第 1 期，第 132 页，2023 年。 [在线]。网址：https://doi.org/10 1038 41377 023 01180 9 。/s - - - Z. Wei，《神经网络中泛化的时空动力学》，2026 [ ] 2602 01651 。在线。网址：https://arxiv.org/abs/。Anthropic，《Claude》，人工智能助手；使用了多个模型版本。[在线]。网址：https://claude.com/product/overview [16] 2026, [17] Anomaly，《OpenCode》，2025 年，开源人工智能编程代理。 [在线]。网址：https://github.com/anomalyco/opencode